Please type a plus sign (+) inside this box → ⊞

PTO/SB/05 (12/97)
Approved for use through 09/30/00. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number

# UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

| | |
|---|---|
| *Attorney Docket No.* | 17201.707 |
| *Total Pages* | |
| *First Named Inventor or Application Identifier* | Philippe Harscoet |
| *Express Mail Label No.* | EL322093492US |

**APPLICATION ELEMENTS**
See MPEP chapter 600 concerning utility patent application contents.

1. [X] Fee Transmittal Form *(Submit an original, and a duplicate for fee processing)*
2. [X] Specification [*Total Pages* 24] *(preferred arrangement set forth below)*
   - Descriptive title of the Invention
   - Cross References to Related Applications
   - Statement Regarding Fed sponsored R&D
   - Reference to Microfiche Appendix
   - Background of the Invention
   - Brief Summary of the Invention
   - Brief Description of the Drawings
   - Detailed Description
   - Claim(s)
   - Abstract of the Disclosure
3. [X] Drawing(s) (37 CFR 1.152) [*Total Sheets* 3 ]
4. [X] Oath or Declaration [*Total Pages* 2 ]
   a. [X] Newly executed (original or copy)
   b. [ ] Copy from a prior application (37 CFR 1.63(d)) *(for continuation/divisional with Box 17 completed)* *[Note Box 5 below]*
      i. [ ] DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
5. [ ] Incorporation By Reference (*useable if Box 4b is checked*) The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.

**Assistant Commissioner for Patents**
*ADDRESS TO:* **Box Patent Application**
**Washington, DC 20231**

6. [ ] Microfiche Computer Program (*Appendix*)
7. Nucleotide and/or Amino Acid Sequence Submission (*if applicable, all necessary*)
   a. [ ] Computer Readable Copy
   b. [ ] Paper Copy (identical to computer copy)
   c. [ ] Statement verifying identify of above copies
8. [X] Assignment Papers (cover sheet & documents(s))
9. [X] 37 CFR 3.73(b) Statement (*when there is an assignee*) [ ] Power of Attorney
10. [ ] English Translation Document (*if applicable*)
11. [ ] Information Disclosure Statement (IDS)PTO-1449 [ ] Copies of IDS Citations
12. [ ] Preliminary Amendment
13. [X] Return Receipt Postcard (MPEP 503) (*Should be specifically itemized*)
14. [X] Small Entity Statement(s) [ ] Statement filed in prior application, Status still proper and desired
15. [ ] Certified Copy of Priority Document(s) (*if foreign priority is claimed*)
16. [ ] Other: .....

17. **If a CONTINUING APPLICATION**, *check appropriate box and supply the requisite information:*

[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No.___/_____

**16. CORRESPONDING ADDRESS**

[ ] *Customer Number of Bar Code Label* **021971** *(Insert Customer No. or Attach bar code label here)* *or* [X] *Correspondence address below*

| | | | | | |
|---|---|---|---|---|---|
| *NAME* | | | | | |
| *ADDRESS* | *WILSON SONSINI GOODRICH & ROSATI* | | | | |
| | *650 Page Mill Road* | | | | |
| *CITY* | *Palo Alto* | *STATE* | *California* | *ZIP CODE* | *94304-1050* |
| *COUNTRY* | *USA* | *TELEPHONE* | *(650) 493-9300* | *FAX* | *(650) 845-5000* |

SUBMITTED BY
Typed or Printed Name George A. Willman Reg. Number 41,378

Signature Date July 2, 1999

DRHC\NRPORTBL\PALIBI\DHI\1050172.1

PTO/SB10 (12/97)
Approved for use through 9/30/00. OMB 0651-0031
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE


| **STATEMENT CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) & 1.27(c)) --SMALL BUSINESS CONCERN** | Docket Number (Optional) **17201.707** |
|---|---|

| | |
|---|---|
| Applicant, Patentee, or Identifier: | Philippe Harscoet |
| Application or Patent No.: | Not Yet Assigned |
| Filed or Issued: | Herewith |
| Title: | METHOD AND SYSTEM FOR DYNAMICALLY LOADING DATA STRUCTURES INTO MEMORY WITH GLOBAL CONSTANT POOL |

I hereby state that I am

[ ] the owner of the small business concern identified below

[x] an official of the small business concern empowered to act on behalf of the concern identified below

| | |
|---|---|
| NAME OF SMALL BUSINESS CONCERN | Planetweb, Inc. |
| ADDRESS OF SMALL BUSINESS CONCERN | 1390 Villa Street, Mountain View, CA 94041 |

I hereby state that the above identified small business concern qualifies as a small business concern as defined in 13 CFR Part 121 for purposes of paying reduced fees to the United States Patent and Trademark Office, in that the number of employees of the concern, including those if its affiliates, does not exceed 500 persons. For purposes of this statement (1) the number of employees of the business concern is the average over the previous fiscal year of the concern of the persons employed on a full-time, part-time, or temporary basis during each of the pay periods of the fiscal year, and (2) concerns are affiliates of each other when either, directly or indirectly, one concern controls or has the power to control the other, or a third party or parties controls or has the power to control both.

I hereby state that rights under contract or law have been conveyed to and remain with the small business concern identified above with regard to the invention described in:

[x] the specification filed herewith with title as listed above

[ ] the application identified above

[ ] the patent identified above

If the rights held by the above identified small business concern are not exclusive, each individual, concern, or organization having rights in the invention must filed separate statements as to their status as small entities, and no rights to the invention are held by any person, other than the inventor, who would not qualify as an independent inventor under 37 CFR 1.9(c) if that person made the invention, or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d), or a nonprofit organization under 37 CFR 1 9(e).

Each person, concern, or organization having any rights in the invention is listed below

[ ] no such person, concern, or organization exists

[ ] each such person, concern, or organization is listed below.

| | |
|---|---|
| NAME OF SMALL BUSINESS CONCERN | |
| ADDRESS OF SMALL BUSINESS CONCERN | |

Separate statements are required from each named person, concern or organization having rights to the invention stating their status as small entities (37 CFR 1.27)

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate (37 CFR 1.28(b)).

| | |
|---|---|
| NAME OF PERSON SIGNING | Kenneth Soohoo |
| TITLE OF PERSON IF OTHER THAN OWNER | Vice President of Engineering & CTO |
| ADDRESS OF PERSON SIGNING | Planetweb, Inc., 1390 Villa Street, Mountain View, CA 94041 |

SIGNATURE _____ DATE 6/28/99

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer. Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

*Application for United States Patent*

*in the name of*

*Philippe Harscoet*

# *METHOD AND SYSTEM FOR DYNAMICALLY LOADING DATA STRUCTURES INTO MEMORY WITH GLOBAL CONSTANT POOL*

Attorney Docket No. 17201-707

*Please direct communications to:*

WILSON SONSINI GOODRICH & ROSATI

650 PAGE MILL ROAD

PALO ALTO, CALIFORNIA 94304-1050

650-493-9300

# METHOD AND SYSTEM FOR DYNAMICALLY LOADING DATA STRUCTURES INTO MEMORY WITH GLOBAL CONSTANT POOL

## CROSS REFERENCE TO RELATED APPLICATION

This application is related to Application No. _____, filed on the same day as the present application, entitled, *Method and System for Global Constant Management*, [Attorney Docket No. 17201.708], which is hereby incorporated herein by reference in its entirety.

## BACKGROUND

Field of the Invention

The invention relates to loading data structures into memory, in particular to loading data structures including instructions and constants.

Description of Related Art

Java is an object oriented programming language, which is often used in a network environment, for example, the Internet. Java's source code is written, and then the source code is compiled into a series of class files. The class files can be stored remotely, for example on a server and then be loaded dynamically when needed on a local system. The class files include bytecode, a set of instructions lower level than the original Java source code, yet higher level than code specific to a particular processor. This helps to allow Java to be particularly suited for the

network environment, so that a variety of different local systems can run the Java programs from a network server. Java classes can be distributed to a variety of different systems, as may be connected to the Internet. For example, when encountering a Web page via a browser, a Java application may be initiated, which would involve the Java class files being loaded via the Internet on to the local system.

A local system that runs the Java classes needs functionality to interpret the Java bytecode. One system that provides such functionality is a Java Virtual Machine. The Java Virtual Machine loads the respective classes from the class files and executes methods as needed. The Java Virtual Machine is typically implemented in software, often associated with a browser, but may also be implemented in hardware.

In order to provide useful network applications to a wide variety of systems, it is desirable to be able to run Java applications on small systems that may not have a large amount of memory. Because such systems are small and may not possess excessive memory, it is helpful to conserve the use of memory used by the Java application, in particular the use of random access memory (read-write memory). One approach is to preload classes into memory, loading into read-only memory the methods and data that do not vary, while loading into random access memory varying data and methods. Such an approach is described in U.S. Patent No.

5,815,718, entitled "*Method And System For Loading Classes In Read-Only Memory*," invented by T. Tock, (hereinafter, "Tock"), which is incorporated herein by reference in its entirety. Classes in Java include typically a number of constants. These constants may require a significant amount of memory on the local system that is running the Java program. The Tock patent indicates that the offline class loader eliminates duplicate constants, in order to combine the constant pools of all the classes in a space efficient manner.

It would be desirable to provide a method and a system which overcome the deficiencies of the prior art.

## SUMMARY OF THE INVENTION

Described here is a method of operating a computer involving data structures in a set of data structures. As unloaded data structures are needed during runtime, a data structure is received from a first memory. The data structure includes one or more sets of instructions and one or more constants. Instructions from the data structure are stored in a first portion of a second memory, which comprises RAM. Constants from the data structure are stored in a second portion of the second memory if only if the respective constant has not been stored in the second portion of the second memory. Indexes in instructions that reference the constants are modified to correspond to the respective locations of the constants in the second portion of the second memory, and at least some instructions from the data structure are read and executed from the RAM.

According to one embodiment of the invention, the data structures comprise classes, and the sets of instructions comprise methods. The classes may comprise Java classes and the methods may comprise Java methods. The constants from the data structure may comprise a constant pool of the data structure. Receiving the data structure from a first memory may comprise receiving the data structure from a server over the Internet.

An embodiment of the invention includes, for classes in a set of classes, as unloaded classes are needed during runtime, receiving a class from a class file, the

class including one or more methods and one or more constants. Instructions from the class are stored in a first portion of a memory. Constants from the class are stored in a second portion of the memory if only if the respective constant has not been stored in the second portion of the memory. Indexes within methods that reference the constants are modified to correspond to the respective locations of the constants in the second portion of the memory, and at least some instructions are executed from the memory from the class before receiving another class from the class file.

An embodiment of the invention includes a computer system including a memory and first logic that, for classes in a set of classes, receives a class from a class file. The class includes one or more methods and one or more constants. The first logic stores instructions from the class in a first portion of the memory and stores constants from the class in a second portion of the memory if only if the respective constant has not been stored in the second portion of the memory. The first logic modifies indexes within methods that reference the constants to correspond to the respective locations of the constants in the second portion of the memory. The computer system includes second logic that executes methods stored in the memory. The memory, the first logic, and the second logic are coupled locally.

## BRIEF DESCRIPTION OF THE DRAWINGS

The invention is illustrated by way of example, and not limitation in the drawings.

FIG. 1 is block diagram of a virtual machine, memory, and system, according to an embodiment of the invention.

FIG. 2 is a flow chart of a method of loading classes into memory, according to an embodiment of the invention.

FIG. 3 is a flow chart of a method of loading classes into memory and recalculating branch addresses, according to an embodiment of the invention.

## DETAILED DESCRIPTION

The following is a description of embodiments of the invention. The embodiments shown help to illustrate the invention. However, it is not intended that the invention be limited to the precise embodiments shown.

Java classes are stored in Java class files. Each class typically includes a constant pool, which contains a set of constants used by the class. Constants from one class are often duplicated in other classes. This duplication can result in a waste of memory if the constants are stored redundantly in the system's memory. The Java Virtual Machine loads classes to be executed from the class files. According to one embodiment of the invention, a global constant pool is created in order to avoid waste of memory for redundantly stored constants. When a class is loaded, the global constant pool is checked to determine whether any of the constants in the class are already in the global constant pool. Such constants are not stored in the constant pool. Constants from the class that are not yet in the global constant pool are stored in the global constant pool. Methods in the class that reference constants in the constant pool of the class are modified so that they reference the correct location in the global constant pool.

Such an approach helps to save memory that would be used by redundant constant entries, in a system where classes are loaded dynamically. For example, a browser used to view a web page may encounter a reference to a Java applet. The

browser loads the classes of the applet and dynamically stores the constants of the respective classes into the global constant pool, storing each constant only once. The methods of the applet are modified to reference the global constant pool. Such an approach is particularly advantageous in Java programs because a Java program is often obtained over a network at the time when the program is needed by the local system. In such a situation, the approach described here is advantageous because it does not require preloading of the classes. This embodiment of the invention is also useful even where classes are obtained from a local source, rather than over a network. In such a situation, this approach has the advantage that classes that are not known to the system until after boot time can be stored efficiently in memory.

Java is often run dynamically. Java classes are often loaded dynamically as they are needed. This loading may take place even over a network. Thus, the Java virtual machine can access classes from a variety of different locations, including local source, for example, on a hard drive, and from remote sources, for example, from a remote server via the Internet. Instructions are stored as bytecode, and the bytecode is executed by a Java Virtual Machine. The bytecode is lower level code than a high level language that is run by an interpreter, and is higher level than a program compiled into machine code. The design of the Java programming language helps to allow Java to run on a number of different local systems, provided

that the local systems have a Java Virtual Machine or equivalent. Thus, Java applications can be distributed widely, for example via the Internet.

FIG. 1 is block diagram of a virtual machine, memory, and system, according to an embodiment of the invention. Source code 110 is provided to compiler 112. Compiler 112 outputs a set of class files 115 which may be stored on server 114. System 124 receives class files 115 via network 116. System 124 includes virtual machine 118, system interface 122, and RAM 120. RAM 120 includes classes 142 and global constant pool 150. Virtual machine 118 includes class loader 138, class parser 140, and execution engine 152.

Network 116 may comprise the Internet, or other network, such as a LAN or an enterprise network. Virtual machine 118 may be implemented in software, hardware or both hardware and software. Virtual machine 118 may comprise a Java Virtual machine or may comprise another system capable of executing the methods in the classes.

The class files 115 on server 114 include a constant pool for each class. For example, class 126 includes constant pool 132, class 128 includes constant pool 134, and class 130 includes constant pool 136. The constants stored within these constant pools within class files may be redundant between the respective class files. The class files 115 also include methods, which have bytecode instructions. Class loader 133 loads respective class files 115 via network 116. Additionally, local storage 160

may include class files, such class file 162, including constant pool 164. Local storage 160 may comprise flash memory, a hard drive, a CD, other memory device, or a combination of these. Class loader loads class files as they are needed dynamically as virtual machine 118 requires. For example, some class files may be loaded initially, and then, as a reference to another class is encountered during execution of the earlier loaded class files, additional class files are loaded.

Class parser 140 parses through class files and stores the data for the class files into RAM 120. RAM 120 may include the heap, such that classes 142 and global constant pool are stored on the heap. In parsing classes, class parser 140 identifies constants from the respective constant pools of classes. Class parser 140 creates a global constant pool 150 in RAM and stores constants from the respective classes into global constant pool 150. If an entry has already been made for the constant, then it is not stored again into the global constant pool. Classes 142 (144, 146, 148) do not have individual constant pools. Thus, redundancies between constants in respective classes are eliminated through the use of the global constant pool 150. Further, classes are parsed dynamically as they are needed by class parser 140, thus eliminating the need for preloading and parsing the classes. Class parser 140 also modifies indexes within methods that refer to constants. In classes received from class file 115, methods index constants that are included within the constant pool of the respective class. For example, class 126 may have a method that

references a constant in its constant pool 132. Now, the index in the method must reference the constant in constant pool 150.

Execution engine 152 supports object oriented code, including the creation of instances and invocation of methods. Execution engine 152 interprets the bytecodes from class 142 as required to execute requested methods. Module 154 in execution engine 152 represents logic in execution engine to support the global constant pool 150. Module 154 causes execution engine to look for constants referenced by methods within classes 142 in the global constant pool 150 rather than in a constant pool of the individual class. Execution engine is coupled to system interface 122, which executes operations requested by execution engine 152. One embodiment, system interface 122 comprises an operating system, which may interface to a file system. And another embodiment to the invention, system interface 122 is simply a file system, so that execution engine 152 is coupled directly to a file system, and the system 124 does not have an operating system. Such a configuration is advantageous where there is a premium on memory. And in this manner, in combination with the use of a global constant pool 150 and lack of an operating system, the need for memory is reduced.

The functionality of the class parser 140 may be combined into class loader 138. In one embodiment of the invention, class loader 138 represents the only class loader in virtual machine 118, and is modified to include the functionality of class

parser 140. In such an embodiment, class loader 138 represents the system class loader. In an another embodiment of the invention, class loader 138, combined with the functionality of class parser 140 is a separate instance, and is independent of the system class loader.

According to one embodiment of the invention, a class is parsed immediately after it is loaded. According to another embodiment of the invention, a class is parsed after a series of classes have been loaded by class loader 138.

In one embodiment of the invention, the structure described here may be applicable to data structures other than Java class files. For example, in one embodiment of the invention, data structures other than class files 115 are stored on a server such as server 114. These data structures include instructions and constants. These data structures may also be stored on local storage 160. When these data structures are loaded onto system 124, the instructions portions of the data structures are stored separately in RAM 120 from the constants 150. Duplicate constants are eliminated such that global constant pool 150 only has one instance of each constant. As instructions from respective data structures are executed, the global constant pool is utilized when constants are referenced.

System 124 in one embodiment of the invention is a small appliance that has a minimum of RAM 120. System 124 may comprise a set top box that is coupled to a television, a game console, a web phone, or other small device. Another

embodiment to the invention, system 124 comprises a computer system, such as a desktop computer. Other embodiments of system 124 are also possible.

FIG. 2 is a flow chart of a method of loading classes into memory, according to an embodiment of the invention. A virtual machine causes classes that were not yet to be loaded and stored into RAM. First the virtual machine is started (block 110). Next, a class is loaded (block 214). The constant pool of the class is isolated (block 216). The constants from the class are stored into the global constant pool (block 218). If a constant is already stored in the global constant pool, then it is not stored again. In this way, space is conserved by avoiding storing duplicate constants in the RAM. An advantage of this approach is that less RAM is needed than would be needed if each class retained its original constant pool.

For methods that index into the constant pool, change the index to an index into the constant pool (block 220). Before the index is changed, members of the constant pool are referenced by an index into the local constant pool of the class. For example, a constant may have an index of '1' in the local constant pool of the class. However, in the global constant pool a number of other constants may have been stored before this particular constant is later stored in the global constant pool. Therefore, the index in the method that references the constant is changed from '1' to properly index into the different global constant pool location. This occurs because

a number of constants from various constant pool have been combined into the global constant pool.

Next, execute requested methods (block 222). If when executing a requested method, a reference to an unloaded class is encountered, then repeat the above (block 224). Otherwise, continue executing requested methods (block 222). An advantage of this method is that constants are stored efficiently in RAM, even in a dynamic environment in which classes are loaded as they are needed by the system. This is a particular advantage when classes are obtained over a network dynamically and one cannot predict which class will be needed for the execution of a particular program, such as a Java applet. For example, an entry in the constant pool may be resolved for the first time when it is used. The resolution includes checking that the item is present in RAM and loading or creating the item if it is not present in RAM. Thus, if the constant pool entry references a class not yet loaded, it can be dynamically loaded and stored in RAM, with the constants being stored into the global constant pool, and methods being modified to index properly into the global constant pool.

FIG. 3 is a flow chart of a method of loading classes into memory and recalculating branch addresses, according to an embodiment of the invention. First the virtual machine is started (block 310). A class is loaded (block 312). The constant pool in the class is isolated (block 314). Constants from the class are stored

into the global constant pool (block 316). If a constant is already stored in the global constant pool, a duplicate entry is not made. For methods indexing into the constant pool, the index is changed to a larger index into the constant pool (block 318). With the global constant pool, constants can be shared between different classes. Here the index into the constant pool is a larger index than the index originally present in the method. This is an advantage if a number of methods are loaded into the virtual machine such that the number of constants is larger than the number possible values for the original index in the method. For example, in Java an 8-bit index may be used to reference into the constant pool and may be present in methods loaded from Java classes. The 8-bit index may be replaced with a 16-bit index in order to allow for a large number of entries in the global constant pool. This change in the size of the index will cause the locations of subsequent bytecode to change. Therefore, branch addresses are recalculated (block 320). For example a branch address may be pointing to an address that was previously used by a bytecode, which is now shifted downward because of the larger indexes now present in the code.

For example, if a branch appears before the shift, the branch value has to reflect the shift as well. Here is the code before the modification:

```
10 branch 14
12 ldc 1  <- 8 bit index
14 ...
```

after relocating the index, it should become like this:

10 branch 15
12 ldcw 0x1234 <- 16 bit index

Thus branch 14 needs to be changed because the location to which it is pointing now contains the second byte of the 16 bit index.

Entries in the exception table are changed to reflect relocation bytecodes due to the use of a larger index. Offsets are adjusted with respect to the new locations of the respective bytecodes.

Next, execute requested methods (block 322). If a reference to an unloaded class is encountered (block 324), then return to loading the class (block 312). Thus, constants are dynamically stored in the global constant pool as classes are dynamically loaded from a class file.

The methods and systems described above also apply in applications other than Java classes. For example, these methods and systems may be applicable to computer systems using other object oriented programming schemes. Alternately, these methods and systems may more generally be applied to the loading of data structures from memory, where the data structures include instructions and constants.

Various embodiments of the invention have been illustrated in the figures and have been described in the corresponding text of this application. This

foregoing description is not intended to limit the invention to the precise forms disclosed. Rather, the invention is to be construed to the full extent allowed by the following claims and their equivalents.

What is claimed is:

## CLAIMS

1. A method of operating a computer, the method comprising:

for data structures in a set of data structures, as unloaded data structures are needed during runtime,

receiving a data structure from a first memory, the data structure including one or more sets of instructions and one or more constants;

storing instructions from the data structure in a first portion of a second memory, the second memory comprising RAM;

storing constants from the data structure in a second portion of the second memory if only if the respective constant has not been stored in the second portion of the second memory,

modifying indexes in instructions that reference the constants to correspond to the respective locations of the constants in the second portion of the second memory, and

reading and executing at least some instructions from the data structure from the RAM.

2. The method of claim 1, wherein the data structures comprise classes.

3. The method of claim 1, wherein the data structures comprise Java classes.

4. The method of claim 1, wherein the sets of instructions comprise methods.

5. The method of claim 1, wherein the sets of instructions comprise Java methods.

6. The method of claim 1, wherein the constants from the data structure comprise a constant pool.

7. The method of claim 1, wherein receiving the data structure from a first memory comprises receiving the data structure from a server over the Internet.

8. The method of claim 1, wherein modifying indexes in instructions includes replacing respective indexes with larger indexes and wherein the method further includes calculating addresses associated with branch instructions.

9. A method of operating a computer, the method comprising:

for classes in a set of classes, as unloaded classes are needed during runtime,

receiving a class from a class file, the class including one or more methods and one or more constants;

storing instructions from the class in a first portion of a memory;

storing constants from the class in a second portion of the memory if only if the respective constant has not been stored in the second portion of the memory,

modifying indexes within methods that reference the constants to correspond to the respective locations of the constants in the second portion of the memory, and

executing from the memory at least some instructions from the class before receiving another class from the class file.

10. The method of claim 9, wherein the classes comprise Java classes.

11. The method of claim 9, wherein the memory comprises RAM.

12. The method of claim 9, wherein receiving the class from a class file comprises receiving the class from a server over the Internet.

13. The method of claim 9, wherein modifying indexes within methods includes replacing respective indexes with larger indexes and wherein the method further includes calculating addresses associated with branch instructions.

14. The method of claim 9, wherein the respective indexes each comprise 8 bits and the larger indexes each comprise 16 bits.

15. The method of claim 9, wherein the constants comprise strings.

16. A computer system comprising:

a memory;

first logic that for classes in a set of classes,

receives a class from a class file, the class including one or more methods and one or more constants;

stores instructions from the class in a first portion of the memory;

stores constants from the class in a second portion of the memory if only if the respective constant has not been stored in the second portion of the memory, and

modifies indexes within methods that reference the constants to correspond to the respective locations of the constants in the second portion of the memory; and

second logic that executes methods stored in the memory;

wherein the memory, the first logic, and the second logic are coupled locally.

17. The computer system of claim 16, wherein the classes comprise Java classes.

18. The computer system of claim 16, wherein the constants from the class comprise a constant pool of the data structure.

19. The computer system of claim 16, wherein the memory comprises RAM.

20. The computer system of claim 16, wherein receiving the class from a class file comprises receiving the class from a server over the Internet.

21. The computer system of claim 16, wherein modifying indexes within methods includes replacing respective indexes with larger indexes and wherein the method further includes calculating addresses associated with branch instructions.

22. The computer system of claim 16, wherein the respective indexes each comprise 8 bits and the larger indexes each comprise 16 bits.

23. The computer system of claim 16, wherein the logic comprises computer readable code means loaded into a RAM.

# METHOD AND SYSTEM FOR DYNAMICALLY LOADING DATA STRUCTURES INTO MEMORY WITH GLOBAL CONSTANT POOL

## ABSTRACT

A method of operating a computer involving data structures in a set of data structures. As unloaded data structures are needed during runtime, a data structure is received from a first memory. The data structure includes one or more sets of instructions and one or more constants. Instructions from the data structure are stored in a first portion of a second memory, which comprises RAM. Constants from the data structure are stored in a second portion of the second memory if only if the respective constant has not been stored in the second portion of the second memory. Indexes in instructions that reference the constants are modified to correspond to the respective locations of the constants in the second portion of the second memory, and at least some instructions from the data structure are read and executed from the RAM. Also described is a computer system including a memory and logic that, for classes in a set of classes, receives a class from a class file and stores constants from the class in a second portion of the memory if only if the respective constant has not been stored in the second portion of the memory.

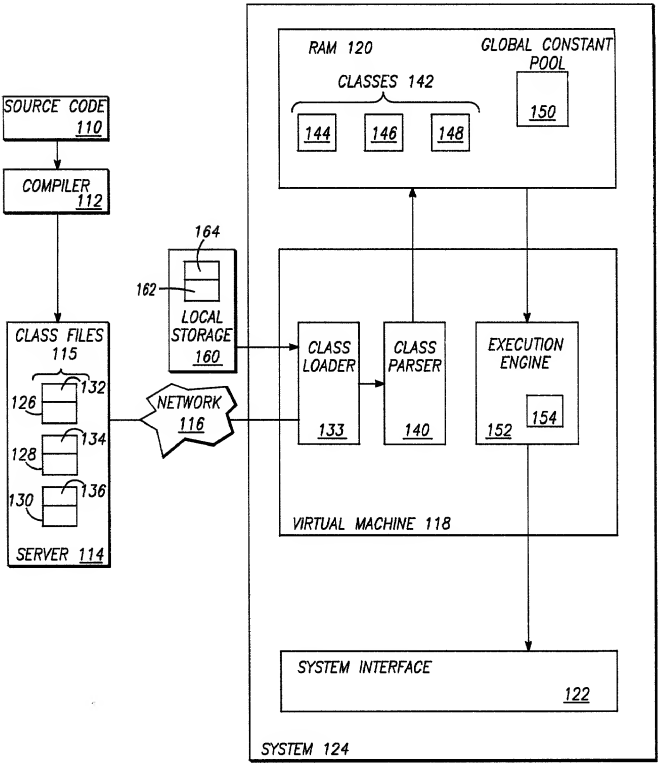
SOURCE CODE
110
COMPILER
112
CLASS FILES
115
132
126
134
128
136
130
SERVER 114
164
162
LOCAL
STORAGE
160
NETWORK
116
RAM 120
GLOBAL CONSTANT
POOL
150
CLASSES 142
144
146
148
CLASS
LOADER
133
CLASS
PARSER
140
EXECUTION
ENGINE
152
154
VIRTUAL MACHINE 118
SYSTEM INTERFACE
122
SYSTEM 124

FIG. – 1

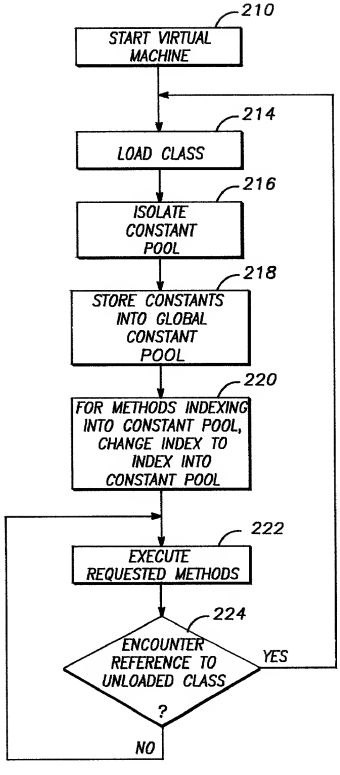
210
START VIRTUAL MACHINE
214
LOAD CLASS
216
ISOLATE CONSTANT POOL
218
STORE CONSTANTS INTO GLOBAL CONSTANT POOL
220
FOR METHODS INDEXING INTO CONSTANT POOL, CHANGE INDEX TO INDEX INTO CONSTANT POOL
222
EXECUTE REQUESTED METHODS
224
ENCOUNTER REFERENCE TO UNLOADED CLASS ?
YES
NO

FIG.—2

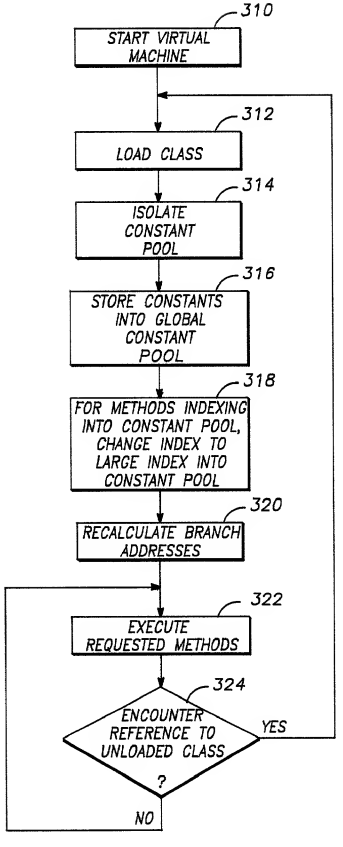
310
START VIRTUAL MACHINE
312
LOAD CLASS
314
ISOLATE CONSTANT POOL
316
STORE CONSTANTS INTO GLOBAL CONSTANT POOL
318
FOR METHODS INDEXING INTO CONSTANT POOL, CHANGE INDEX TO LARGE INDEX INTO CONSTANT POOL
320
RECALCULATE BRANCH ADDRESSES
322
EXECUTE REQUESTED METHODS
324
ENCOUNTER REFERENCE TO UNLOADED CLASS ?
YES
NO

FIG.—3

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

| | |
|---|---|
| In re Application of | ) PATENT APPLICATION |
| Inventor: Philippe Harscoet | ) |
| Application No.: Not Yet Assigned | ) |
| Filed: Herewith | ) |
| Title: **METHOD AND SYSTEM FOR DYNAMICALLY LOADING DATA STRUCTURES INTO MEMORY WITH GLOBAL CONSTANT POOL** | ) |

**DECLARATION FOR PATENT APPLICATION**

As a below named inventor, I hereby declare that my residence, post office address and citizenship are as stated below next to my name; I believe that I am the original, first and sole inventor (if one name is listed below), first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

**METHOD AND SYSTEM FOR DYNAMICALLY LOADING DATA STRUCTURES INTO MEMORY WITH GLOBAL CONSTANT POOL**

the specification of which (check applicable ones):

__x__ is attached hereto;

____ was filed with the above-identified "Filed" date and "SC/Serial No."

_____ was amended on (or amended through) _____.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose information which is material to the examination of the application in accordance with Title 37, Code of Federal Regulations, §1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate as indicated below and have also identified below any foreign application for patent or inventor's certificate on this invention having a filing date before that of the application on which priority is claimed:

| Prior Foreign Application(s) | | | Priority Claimed | |
|---|---|---|---|---|
| (Number) | (Country) | (Day/Month/Year Filed) | Yes | No |

I hereby claim the benefit under Title 35, United States Code, §120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, §112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulation, §1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

| (Application Serial No.) | (Filing Date) | (Patented, Pending, Abandoned) |
|---|---|---|

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under §1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

(1) Full name of sole or first inventor: Philippe Harscoet

(1) Residence: 2222 Duvall Court, Santa Clara, California 95054

(1) Post Office Address: same

(1) Citizenship: French

(1) Inventor's signature: 

(1) Date: 6/28/99

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

**PATENT APPLICATION**

IN RE PATENT APPLICATION OF )
)
Philippe Harscoet )
)
)
Application No.: Not Yet Assigned )
)
Filing Date: Herewith )
)
Title: **METHOD AND SYSTEM FOR DYNAMICALLY** )
**LOADING DATA STRUCTURES INTO MEMORY** )
**WITH GLOBAL CONSTANT POOL** )

**POWER OF ATTORNEY BY ASSIGNEE**
**TO EXCLUSION OF INVENTOR UNDER 37 C.F.R. § 3.71**
**WITH REVOCATION OF PRIOR POWERS**

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

The undersigned ASSIGNEE of the entire interest in the above-identified application for letters patent hereby appoints:

| | |
|---|---|
| Paul Davis | 29,294 |
| John J. Bruckner | 35,816 |
| David J. Weitz | 38,362 |
| Kent R. Richardson | 39,443 |
| David J. Abraham | 39,554 |
| U.P. Peter Eng | 39,666 |
| Henry J. Groth | 39,696 |
| George A. Willman | 41,378 |
| Van Mahamedi | 42,828 |
| Richard L. Gregory, Jr. | 42,607 |
| Travis L. Dodd | 42,491 |
| Barbara Courtney | 42,442 |
| Jinntung Su | 42,174 |
| Shantanu Basu | 43,318 |



C:\NRPORTBL\PALIB1\DH1\1034005.1
Attorney Docket No.: 17201.708

to prosecute this application and transact all business in the United States Patent and Trademark Office in connection therewith and hereby revokes all prior powers of attorney; said appointment to be to the exclusion of the inventors and the inventors' attorneys in accordance with the provisions of 37 C.F.R. § 3.71.

The following evidentiary documents establish a chain of title from the original owner to the Assignee:

_X_ a copy of an Assignment attached hereto, which Assignment has been (or is herewith) forwarded to the Patent and Trademark Office for recording; or

__ the Assignment recorded on _____ at reel ___, frames ___ - ___.

Pursuant to 37 C.F.R. § 3.73(b) the undersigned Assignee hereby states that evidentiary documents have been reviewed and hereby certifies that, to the best of ASSIGNEE's knowledge and belief, title is in the identified ASSIGNEE.

Direct all telephone calls to George A. Willman, (650) 493-9300.

Address all correspondence to:

George A. Willman
WILSON SONSINI GOODRICH & ROSATI
650 Page Mill Road
Palo Alto, California 94304-1050

ASSIGNEE: Planetweb, Inc.

Name: _____
(Signature)

Name: Kenneth Soohoo
(Print or Type)

Title: Vice President of Engineering, CTO

Date: 6/28/99



C:\NRPORTBL\PALIB1\DH1\1034005.1
Attorney Docket No.: 17201.708